KERMIT USERS GUIDE

Fourth Edition

Frank da Cruz, Daphne Tzoar, Bill Catchings[1]

Columbia University Center for Computing Activities
New York, New York 10027

5 April 1985

---

[1]Present address: Lehman Bros, Kuhn & Loeb, NYC

Preface to the 4th Edition

The Kermits are coming home to roost. Recipients of earlier Kermit distributions have sent back their own contributions. The 4th edition includes these, plus additional work done at Columbia (where no attribution is made below, the work was done at Columbia):

* KERMIT-32 was written for the VAX-11 running VMS by Bob McQueen at Stevens Institute of Technology. KERMIT-32 can act as a server.

* KERMIT-65 was written for the Apple II 6502-based DOS system by Antonino Mione at Stevens Institute of Technology.

* KERMIT-10 now has CONNECT command, added by Vanya Cooper at Pima Community College in Tucson, Arizona.

* KERMIT-20 has various new SET options, load-dependent timeouts, improved support for local operation (comforting messages on the screen to show the file and packet traffic), expanded help and information facilities, and the ability to issue commands to a Kermit Server (KERMIT-20 could already act as a Kermit Server itself), expanded debugging options, etc.

* Kermit-86 can now issue commands to a server, and can do wildcard sends. Z100 support was added to Kermit-86 at Stevens Institute of Technology.

* An adaptation of Kermit-80 was done by DEC for the Rainbow-100. This is not the "real" Rainbow Kermit, but a stopgap. It will only work at speeds up to 1200 baud. A similar adaptation was done for the DECmate-II with CP/M.

* A "generic" version of Kermit-80 has been done, which should work on any CP/M-80 system that implements the CP/M IOBYTE.

* An implementation of Kermit-80 was done for the TRS-80 II and for systems running CP/M-Plus (3.0) by Bruce Tanner at Cerritos College in Norwalk, California. Another for the Osborne 1 by Charles Bacon at the National Institutes of Health. And another for the Telcon Zorba by Nick Bush at Stevens.

* A version for the Kaypro II was written by Chad Pearce at Miller, Mason & Dickenson Inc, Philadelphia PA.

* The SYMBOL cross assembler was "reTOPS10ized" at several sites, but it has been dropped from the KERMIT distribution altogether in favor of MAC80, an 8080-specific cross assembler that assembles KERMIT-80 much faster than SYMBOL could do, and produces a smaller hex file in the bargain. SYMBOL was written in SAIL, which is never quite the same on any two TOPS-10 or TOPS-20 systems; MAC80 is written in MACRO-10, and doesn't require any special runtime support, and runs with no problems under either TOPS-10 or TOPS-20. MAC80 was contributed by Bruce Tanner.

* The RT-11 OMSI Pascal implementation contributed earlier by Philip Murton of the University of Toronto was fixed up by Michael Weisberg of Mt. Sinai Hospital in NYC to be self contained and complete, and actually brought up on real RT-11 systems (the previously distributed version was incomplete).

* The Columbia CS Department UNIX Kermit was updated to distinguish between "image mode" file transfers (e.g. UNIX-to-UNIX), and ordinary transfers (in which newlines are mapped to CRLFs).

Preface to the 3rd Edition

For the 3rd edition, the Kermit manual has been reorganized to improve clarity and to prune away a lot of redundant material. The wording and examples have become less oriented to Columbia, and less towards programmers. Much new material has been added as a result of new implementations of Kermit done at Columbia and elsewhere, notably for TOPS-10, UNIX, and the IBM PC. Other additions describe techniques for bootstrapping Kermit onto new systems, complications arising from system bytesize mismatches, and a new simplified way of running Kermit. The section describing the Kermit protocol has been moved to a new document, the Kermit Protocol Manual.

Some material has been removed that described how to use certain systems. Due to the proliferation of systems supporting KERMIT, it is no longer practical to describe how to use each system. It is now assumed that the reader knows how to log in and run programs on the systems involved, how to start a micro, insert a floppy disk, and so forth.

Ordering Information

The KERMIT software is free and available to all. Columbia University, however, is not set up to distribute free software on the scale required for KERMIT. Therefore, to defray our costs for media, printing, postage, labor, and computing resources, we must request a moderate donation from sites that request KERMIT directly from Columbia. The schedule is as follows:

Complete KERMIT Distribution (Tape, Users Guide, and Protocol Manual)	$100.00
Printed Documents (Users Guide, Protocol Manual, or Any Source Listing)	$5.00 each

Other sites are free to redistribute KERMIT on their own terms, and are encouraged to do so, with the following stipulations: KERMIT should not be sold for profit; credit should be given where it is due; and new material should be sent back to Columbia University at the address below so that we can maintain a definitive and comprehensive set of KERMIT implementations for further distribution.

To order KERMIT from Columbia University, send a letter requesting either:

(a) The manuals or source listings you desire (specify each one), or

(b) A 9-track magnetic tape in one of the following formats.

System	Tape Format	Densities
TOPS-10	BACKUP/Interchange, Unlabeled	800, 1600
TOPS-20	DUMPER, Unlabeled	800, 1600
IBM VM/CMS	EBCDIC, CMS Format	1600, 6250
	or EBCDIC, OS Standard Label	1600, 6250
Other	ASCII, ANSI Label, Format "D"	800, 1600

(Specify system, format, and density.) One copy of each manual will be included with the tape. We will supply the tape, packaging, and postage.

We can only make tapes in the formats listed above. We cannot produce floppy disks; bootstrapping procedures are provided to allow the microcomputer versions to be downloaded from the mainframe for which the tape is produced. The tape includes all source programs, documentation, and, when practical, binaries or hex.

Send your letter to:

KERMIT Distribution
Columbia University Center for Computing Activities
7th Floor, Watson Laboratory
612 West 115th Street
New York, N.Y. 10025

Please list the machines and operating systems you expect to run KERMIT on, specify the tape format or the listings desired, and mention whether there are additional systems for which you require KERMIT or if you might be interested in attempting your own im-

plementation. Make checks payable to Columbia University Center for Computing Activities.

KERMIT is also available to users of the BITNET network via a server at host CUVMA. BITNET users may type "SMSG RSCS MSG CUVMA KERMSRV HELP" for further information. Additional network and user group distribution methods are under consideration. Suggestions would be welcome.

No warranty of the software nor of the accuracy of the documentation surrounding it is expressed or implied, and neither the authors nor Columbia University acknowledge any liability resulting from program or documentation errors.

1. Introduction

Everyone wants to get computers talking to one another. There are many ways to do this, and most of them are very expensive. But there is one way that is cheap and relatively easy: connect the two computers through their terminal (TTY) ports, tricking one computer (or both) into believing that the other is a terminal. This can be expected to work because the standard for connecting computers to terminals is almost universally followed, in both hardware (plug and signal: EIA RS-232) and software (character code: ASCII). Once two computers (hosts) are connected in this way, cooperating programs can be run on each host to achieve the desired communication by means of a communication protocol.

Why is a protocol necessary at all? Two major problems occur when you try to connect two computers via TTY line:

1. Noise. Data can become garbled in transmission because of electrical interference, cosmic rays, or rodents chewing on the wires. The longer a wire, the more noise you can expect. The noise corrupts the data, sometimes in subtle ways which you may not notice until it's too late.

2. Synchronization. One computer may be faster, or have larger buffers, than the other. If one computer can send data faster than the other can take it in, then some kind of flow control mechanism, either in the data stream (XON/XOFF) or in parallel to the data (modem control signals), can be employed to prevent data overruns, but no two computers can be assumed to honor the same conventions for flow control, or indeed to do it at all. Even when flow control is being done, those signals are themselves are subject to noise corruption.

To prevent corruption of data and to synchronize communication, cooperating computers can send control information to one another at the same time that they are transferring data. This intermingling of control information with data, and the resulting actions, constitute a "protocol".

KERMIT is a simple example of such a protocol. It is specifically designed for transfer of files over ordinary serial communication lines. KERMIT is not necessarily better than many other terminal-oriented file transfer protocols but it is free, it is well documented, and it has been implemented compatibly on a variety of microcomputers and mainframes.

KERMIT transfers data by encapsulating it in "packets" of control information. This information includes a synchronization marker, a packet number to allow detection of lost packets, a length indicator, and a "checksum" to allow verification of the data. Lost or corrupt packets are detected, and retransmission is requested. In addition, various special control packets allow cooperating KERMITs to connect and disconnect from each other and to exchange various kinds of information. Very few assumptions are made about the capabilities of either computer, so the KERMIT protocol can work between many different kinds of systems.

It should be noted that the checksum technique used by Kermit is not foolproof. There

are conditions under which bad data can still generate a valid checksum. In practice, however, the protocol is quite reliable.

Acknowledgements to the Stanford DIALNET project (John McCarthy and Mark Crispin), Peter Hurley of DEC (the "Hurley Protocol" described in the only known issue of the DECsystem-10 Journal of Applications and Research), the Stanford University Medical Center MAINSAIL project (TTYFTP) the University of Utah CS Department SMALL-FTP developers, and to the authors of the File Transfer Protocol section of the ARPANET Protocols Handbook.

The original design was done by Bill Catchings, Frank da Cruz, and Vace Kundakci of the Columbia University Computer Center, in consultation with some potential users, notably Peter Brooks. Bill wrote the initial implementations: one for TOPS-20 and one for a Z80 CP/M system, which has subsequently been adapted for a number of other CP/M systems. Bill also wrote the C kernel, which has been used as the basis of other implementations, particularly the UNIX implementation of Bob Cattani and Chris Maio of our CS Department. Daphne Tzoar of Columbia wrote and documented the IBM VM/CMS and the IBM PC implementations. Other contributors are noted appropriately throughout this manual.

2. How to Use KERMIT

KERMIT is a protocol for reliable file transfer between computers over the ordinary serial telecommunication lines that are used to connect terminals to computers. The mechanics of using KERMIT to get a file transferred can be confusing until you get the hang of it. A little background material might make the process a bit easier to understand.

KERMIT is probably the cheapest way to put two computers into communication. The required hardware is usually already available, the software is free, and all components run as ordinary user programs, with no system modifications. This is in sharp contrast to a communication network, where there are dedicated high-speed communications channels and drivers, expensive software, and so forth. The network provides more services than KERMIT, usually at higher speed, and with greater convenience, because the network is usually part of the system. When a network is not available, KERMIT can fill in. But since KERMIT is not integrated with any particular system, but rather grafted on top of many different systems, it requires some extra work from those who use it.

2.1. The KERMIT Program

KERMIT embodies a set of rules for transferring files reliably between computers. In general, one computer is a large system (a host, for instance a timesharing system with many terminals), and the other is a personal computer (PC)[2]. The host believes that the PC is an ordinary terminal. In order for the KERMIT protocol to occur, a KERMIT program must be running on each end of the communication line -- one on the host, one on the PC.

The two Kermit programs exchange messages in a special language all their own, the Kermit protocol. The dialog runs something like, "Hi! I'm going to be sending files to you. When you send messages to me, please don't make them more than 80 characters long, and if you don't hear anything from me for 15 seconds, wake me up, OK?" "OK." "Now, here comes a file called FOO.TXT, OK?" "OK." "Here's the first piece..." "Got it." "Good! Here's the second piece..." "Oops, sorry, the second piece was inconsistent." "Well, then here it is again..." Et cetera. You don't see any of this. It's all packed into a very concise code which the two Kermits can understand; they do all the worrying about transmission, error checking, character set translation, and so forth. Each message is called a packet, and each packet is in a special format that all Kermits can understand.

---

[2] Host-to-host and PC-to-PC connections are also possible.

2.2. Talking to Two Computers at Once

Your task is just to get the two Kermits started. The confusion arises because you have to use a single keyboard and screen to talk to two different computers, two different programs. Let's talk about a common case: you are sitting at a personal computer (PC[3]), which has a serial communication port. The serial port is connected to a host computer using, say, a dialup modem[4].

Normally, when you use your PC, you are "talking" directly to it; your commands are interpreted directly by the PC's operating system (CP/M, MS-DOS, UNIX, whatever), or by some program that runs on the PC (an editor, a text formatter, space invaders...). The version of Kermit on your PC is a program like any other, but it has a special ability to either interpret your commands directly, like other programs, or to pass everything you type through to the host. When you tell Kermit to CONNECT, it sends every character you type out the serial port, and it will put every character that comes in the serial port onto the screen. This is called virtual terminal service -- one computer acts "virtually" as though it were a terminal on another. You are now "talking" to the host, and the PC is ignoring you.

Kermit, like most programs, has a prompt. The prompt is a symbol it types on the left margin to indicate that it is ready for you to type a command. Kermit's prompt is normally "Kermit-xx>". The xx identifies the implementation of Kermit; the Kermit that runs on the DEC-20 is called "Kermit-20" and its prompt is "Kermit-20>"; the Kermit that runs on Z80 and 8080-based microcomputers is called "Kermit-80" and its prompt is "Kermit-80>"; the Kermit on the IBM PC is "Kermit-86"[5], and so forth. If you become confused about who you are talking to, the prompt should provide a clue. In addition, most Kermits print an informative message like

```
[Connecting to remote host, type CTRL-]C to return]
```

when you CONNECT, and type another message like

```
[Connection closed, back at PC]
```

when you return.

Having "connected" to the host, there must be a way for you to get back to the PC. This is accomplished by an escape sequence. As Kermit passes your characters through to the host, it checks each one to see if it's a special predefined escape character. When the PC sees this character, it stops ignoring you -- you are once again "talking" to the

---

[3]The terms PC, micro, microcomputer, and workstation will all be used loosely in this document to denote a single-user system.

[4]The actual means of connection isn't important in this case -- it also could be a direct line to the host, some kind of switched line, etc.

[5]Although the processor in the IBM PC is an 8088, it is programmed as though it were an 8086.

PC, not the host. The escape character is normally chosen to be one that you will not need to type while talking to the host, and one that is hard to type by accident -- it's usually a control character, such as Control-], which is accomplished by holding down the key marked CTRL or CONTROL and typing the indicated character (in this case, a right bracket "]"). The CTRL key works just like a SHIFT key. Control characters are written either as CTRL-A or ^A, where A is the character to be typed while holding down CTRL.

2.3. Transferring a File

To transfer a file, you must first get the attention of the PC's operating system. This is normally done by starting the PC, possibly inserting your system floppy disk first. Once you're at command level on your PC, you run Kermit. Then you tell Kermit to CONNECT you to the host. Now you're talking to the host -- at this point you must log in, and then run Kermit on the host.

Now you have a Kermit on each end of the wire. The next step is to tell each Kermit what to do. Suppose you want to transfer a file from the host to the PC; you would first tell the host Kermit to SEND the file, then "escape" back to the PC Kermit and tell it to receive the file. The transfer begins -- all you have to do now is sit back and watch. The PC Kermit will continuously show packet and retry counts on your screen, and will notify you when the transfer is complete.

The desired file is now on your PC's disk. The Kermit protocol has ensured that the file arrived correctly and completely. Now you must clean up after yourself: CONNECT back to the host, exit from Kermit on the host, log out from the host, "escape" back to PC Kermit and exit from it. Now you can do whatever you had planned for your file -- edit it, print it on your PC printer, etc.

The KERMIT protocol, and most Kermit programs, allow you to send a file reliably from the host to the PC, from the PC to the host, from host to host, or PC to PC, usually without any special regard for the nature of the particular machines involved. Most implementations also allow files to be sent in groups, with a single command, such as "Send all my Fortran files!" The scenario for each of these is always the same as above -- only the details of how to establish the actual connection differ.

KERMIT works best with "printable" files -- files composed only of letters, digits, punctuation marks, carriage returns, tabs, and so forth -- since these can be represented on almost any kind of computer. KERMIT is also able to transfer "binary" files -- files such as executable programs -- composed of arbitrary bit patterns, but binary files normally are meaningful only to the kind of computer on which they are generated. Nevertheless, KERMIT can usually move such files from system A to system B (where they are not much use) and back to system A in their original condition, although in some cases some special care must be taken to accomplish this.

Now that we have a basic understanding of what KERMIT does and how it works, let's look at some more concrete examples. First you need to know what the basic Kermit commands are.

2.4. Basic KERMIT Commands

These are generic descriptions of the most basic Kermit commands. Detailed descriptions for specific implementations of Kermit will come later. An underscored command is standard and should be found (in some form) in any implementation of Kermit.

In these descriptions, local refers to the system that you are directly connected to, remote refers to the system to which you are CONNECTed via Kermit. Commands may take one or more operands on the same line, and are terminated by a carriage return.

SEND filespec — Send the file or file group specified by filespec from this Kermit to the other. The name of each file is passed to the other Kermit in a special control packet, so it can be stored there with the same name. A file group is usually specified by including "wildcard" characters like "*" in the file specification. Examples:

```
send foo.txt
send *.for
```

Some implementations of Kermit do not support transfer of file groups; these versions would require a separate SEND command for each file to be transferred.

RECEIVE — Receive a file or file group from the other Kermit. If an incoming file name is not legal, then attempt to transform it to a similar legal name, e.g. by deleting illegal or excessive characters. The name thus formed cannot be guaranteed to be unique, in which case previously existing files could be overwritten. Some versions of Kermit attempt to prevent this by warning you of filename collisions and taking, or allowing for, evasive action.

CONNECT — Make a "virtual terminal" connection to the remote system. On a PC or micro, this usually means to send all keyboard input out the serial port, and display all input from the serial port on the screen. Mainframe KERMITs that have this command behave similarly; they send the characters typed at the terminal to a specified TTY line and pass all input from that TTY line to the terminal. KERMIT connections are most commonly initiated from the the small computer. To "escape" from a virtual terminal connection, type Kermit's escape character (e.g. CTRL-], control-rightbracket), followed by the letter "C" for "Close Connection"

SET — Establish various parameters, such as communication line number, CONNECT escape character, etc. See the description of the appropriate version of KERMIT in Chapter -IMPLEMENTATIONS.

SHOW — Display the values of SET options.

HELP — Type a summary of KERMIT commands and what they do.

EXIT — Exit from KERMIT back to the host operating system.

? — Typed anywhere within a KERMIT command: List the commands, options, or operands that are possible at this point. This command may or may not require a carriage return, depending on the host operating system.

2.5. Real Examples

Kermit is used most commonly in several ways: a user sitting at a PC (personal computer, microcomputer, workstation) which is connected to a larger host computer; a user logged in to a host computer which is connected to another host; two users, each sitting at a PC.

2.5.1. PC to Host

In this example, the user is sitting at an IBM Personal Computer (PC), which is connected through its serial port to a DECSYSTEM-20 host computer. The IBM PC is local, the DEC-20 is remote. This example will also apply almost literally to any other microcomputer implementation of Kermit.

You have started up your PC and have the Kermit program on your disk. Begin by running Kermit on the PC. Use Kermit's CONNECT command to connect to the DEC-20. The PC will now behave as a terminal to the DEC-20. In fact, the PC emulates the popular DEC VT52 terminal, so it is desirable to tell the DEC-20 that your terminal is a VT52. Login on the DEC-20 and run Kermit there: Here is an example of this procedure with commands you type underlined:

```
A>kermit                ! Run Kermit on the PC.[6]
Kermit V1.2

Kermit-86>              ! This is the Kermit prompt for the PC.
Kermit-86>connect       ! Connect to the DEC-20.
[Connecting to host. Type CTRL-]C to return to PC.]

                        ! You are now connected to the DEC-20.
CU20B                   ! The system prints its herald.
@terminal vt52          ! Set your terminal type (optional).
@login my-id password                    ! Login using normal

(At this point, the DEC-20 prints various messages.)

@kermit                 ! Run Kermit on the DEC-20.
Kermit-20>              ! This is Kermit-20's prompt.
```

---

[6] Everthing from a "!" mark to the end of line is a comment, not system typeout or part of a command.

You are now ready to transfer files between the two machines.

2.5.1.1. Receiving from the Host

The following example illustrates how to send files from the DEC-20 to the PC. Note the use of the "*" wildcard character to denote a file group.

```
Kermit-20>send *.for ! Send all FORTRAN files.
^]c                  ! Now return back to the PC by
                     ! typing the escape sequence, in this ca
                     ! ^]C (Control-] followed by "C")
[Back at PC.]
Kermit-86>receive    ! Tell the PC files are coming.
```

If you take more than about 5 seconds to get back to Kermit-86 and issue the RECEIVE command, the first packets from Kermit-20 may arrive prematurely and appear as garbage on your screen, but no harm will be done because the packet will be retransmitted by the DEC-20 until the PC acknowledges it.

Once the connection is established, the PC will show you what is happening -- it first clears the screen and waits for incoming packets; as packets arrive, the current file name and packet number will be continuously displayed on the screen. When the PC's "Kermit-86>" prompt returns to your screen, the transfer is done.

When the transfer is complete, you must CONNECT back to the DEC-20 host, EXIT from Kermit there, logout from the DEC-20, and return to the PC as you did previously.

```
Kermit-86>connect    ! Get back to the DEC-20.
[Connecting to host. Type CTRL-]C to return to PC.]
Kermit-20>           ! Here we are.
Kermit-20>exit       ! Get out of Kermit-20.
@logout              ! Logout from the DEC-20.

Logged out Job 55, User MY-ID, Account MY-ACCOUNT, TTY 146,
  at 24-Jan-83 15:18:56,  Used 0:00:17 in 0:21:55

^]c                  ! Now "escape" back to the PC,
[Back at PC.]
Kermit-86>exit       ! and exit from the PC's Kermit.
```

The files you transferred should now be on your PC disk.

2.5.1.2. Sending to the Host

To send files from the PC to the DEC-20, follow a similar procedure. First follow the instructions in the previous section to log in to the DEC-20 through the PC. Then in response to the host Kermit's "Kermit-20>" prompt you type RECEIVE rather than SEND. Now escape back to the PC and use the SEND command to send the local PC files to DEC-20 host. The PC will show you the progress of the transmission on its screen.

When the "Kermit-86>" prompt indicates that the transmission is complete you should follow the procedure shown above to logout from the DEC-20 host, except that you may first wish to confirm that the files have been stored correctly in your directory on the DEC-20.

2.5.2. Host to Host

This section describes use of Kermit between two hosts. A "host" is considered to be a large or multi-user system, whose distinguishing characteristic is that it has multiple terminals. Use of Kermit for host-to-host file transfers differs from the PC-to-host case in that the line your terminal is connected to is not the same as the line over which the data is being transferred, and that some special commands may have to be issued to allow one Kermit to conform to unusual requirements of the other host.

In this example, you are already logged in to a DEC-20, and you use an autodialer to connect to an IBM 370-style system running VM/CMS through DEC-20 TTY port 12. The autodialer, in this example, is invoked from program called DIAL (idealized here, for simplification), to which you merely supply the phone number.

```
@dial 765-4321/baud:300
765-4321, baud 300
[confirm]
Dialing your number, please hold...
Your party waiting is on TTY12:
@
```

Other methods exist for connecting two hosts with a serial line. Dedicated hookups can be made simply by running an EIA cable between TTY ports on the two systems.[7] For connecting to remote systems when no autodialer is available, a manual dialup connection is also possible, but tricky.[8] If you have a microcomputer that supports KERMIT, you may find it easier to first transfer from host A to the micro, then from the micro to host

---

[7] Such a connection, by the way, requires the receive and transmit leads be swapped in one of the RS-232 connectors; this is called a "null modem" cable.

[8] Here's one way: log in on port x on your system, and assign another port, y, to which you have physical access. Unplug the terminal from port y, and connect the terminal to a dialup modem. Dial up the remote computer and log in on it. Now, using a null modem cable, connect the modem directly to port y. Go back to your terminal on port x, run Kermit from it, and CONNECT to port y.

B.

The following procedure would be the same in any case, once a connection is made. Note that Kermit-20 accomplishes remote terminal connection by running a program called TTLINK in an inferior fork.

```
@
@kermit                ! Run Kermit on the DEC-20.
Kermit-20>set ibm-flag                    ! Turn on special ha
Kermit-20>set line (to tty) 12 ! Indicate the line we'll use.
Kermit-20>connect    ! And connect to it.
[TTLINK: Connecting to remote host over TTY12:, type <CTRL-Y>

VM/370 ONLINE          ! The IBM system prints its herald.

.login myuserid XXXXXX                    ! Login to the IBM s

LOGON AT 20:49:21 EST THURSDAY 01/20/83
CUVMB SP/CMS PUT 8210 01/19/83
.
.kermit
KERMIT-CMS>.send profile exec ! Send a file.
^Yc                    ! TTLINK's escape sequence typed here.
[TTLINK: Connection Closed. Back at DEC-20.]
Kermit-20>receive    ! Tell Kermit-20 to RECEIVE.
```

The transfer takes place now; Kermit-20 will print the names of incoming files, followed by dots or percents to indicate the packet traffic (a dot for every 5 packets successfully transferred, a percent for every timeout or retransmission). It is complete when when you see "[OK]" and the Kermit-20 prompt next appears. At that point we connect back to the remote IBM system, exit from the remote Kermit and log out.

```
.
PROFILE.EXEC.1 ..%%.[OK]
Kermit-20>connect    ! Get back to IBM and clean up.
[TTLINK: Connecting to remote host over TTY12:, type <CTRL-Y>

KERMIT-CMS>.
KERMIT-CMS>.exit
R;

.
SP/CMS

.logout
```

```
CONNECT= 00:03:01 VIRTCPU= 000:00.12 TOTCPU= 000:00.60
LOGOFF AT 20:52:24 EST THURSDAY 01/20/83

^Yc                    ! Type Kermit-20's escape sequence
[TTLINK: Connection Closed. Back at DEC-20.]
Kermit-20>exit         ! All done with Kermit.
```

That's the whole procedure. The file is in your DEC-20 directory, completely readable, as PROFILE.EXEC -- note that KERMIT-CMS translated from the IBM EBCDIC character encoding into standard ASCII, and converted the space between the file name and file type to a dot.

To send a file from the local host to the remote host, we would merely have reversed the SEND and RECEIVE commands in the example above.

2.5.3. Micro to Micro

Kermit also works between personal computers (microcomputers, workstations). The difference here is that commands are typed on two keyboards, rather than a single one. This is because a personal computer normally only accepts commands from its own keyboard. If one PC Kermit CONNECTs to another, there will normally be no program on the other side to listen.

Making the physical connection between two micros is tricky. If the two units are in close proximity[9], you can connect their serial ports with a null modem cable. However, different micros have different requirements -- some may want a male connector on their serial port, others a female; many require that certain of the RS-232 signals be held high or low by wiring certain of the pins in the connector together[10]. In any case, you must also make sure the port speeds are the same at both ends.

Connections at longer distances can be made via dialup, providing the required modems are available (one side needs autoanswer capability), or using any kind of dedicated or switched circuit that may be available -- PBX, port contention unit, anything you can plug an EIA connector into.

In this example, a DEC VT180 "Robin" CP/M microcomputer is connected to an Intertec "SuperBrain" CP/M micro, using a female-to-male null modem cable. Establishing the physical connection is the hard part. The connection can be tested by running Kermit and issuing the CONNECT command on both ends: typein from each micro should appear on the screen of the other.

---

[9] Why would you want to run Kermit between two PCs that are next to each other? One good reason is that if they are different models, their floppy disks are probably incompatible.

[10] For instance, some micros want DTR (Data Terminal Ready, pin 20) to be held high, and this might be accomplished by connecting it to CTS (Clear To Send, pin 5). See EIA Standard RS-232-C, and the appropriate manuals for your micro.

Suppose you want to send a file FOO.HEX from the Robin to the SuperBrain. Proceed as follows:

1. Run Kermit on the SuperBrain, and give the RECEIVE command:

```
A>kermit
CUCCA SuperBrain Kermit-80 - V3.2
Kermit-80>receive
```

2. Run Kermit on the Robin, and give the SEND command for FOO.HEX.

```
A>kermit
CUCCA/DEC VT18X Kermit-80 - V3.2
Kermit-80>send foo.hex
```

3. Watch the packets fly. When you get a Kermit-80> prompt, the transfer is done, and you can EXIT from both Kermits.

The key point is to start the receiving end first -- most microcomputer Kermits do not include a timeout facility, and if the receiver is not ready to receive when the sender first sends, there will be a protocol deadlock.

2.6. Another Way -- The KERMIT Server

So far, we have been describing Version 1 of the KERMIT protocol. This edition of the KERMIT manual now presents Version 2, which includes the concept of a Kermit server. A KERMIT server is a Kermit program that does not interact directly with the user, but only with another Kermit. You do not type commands to a Kermit server, you merely start it at one end of the connection, and then type all further commands at the other end.

Not all implementations of Kermit can be servers, and not all know how to talk to servers -- yet. The server is run on the remote computer, which would normally be a large host, such as the DEC-20. You must still connect to the DEC-20 to log in and start the server, but you no longer have to tell one side to SEND and the other to RECEIVE, nor must you connect back to the remote side to clean up when you're done. Using the server, you can do as many send and receive operations as you like without ever having to connect back to the remote host.

A Kermit server is just a Kermit program running in a special mode. It acts exactly like ordinary Kermit does after you give it a RECEIVE command -- it waits for a message from the other Kermit, but in this case the message is a command telling whether to send or to receive, and if to send, which file(s) to send. After escaping back to the local system, you can give as many SEND and RECEIVE commands as you like, and when you're finished transferring files, you can give the BYE command, which sends a message to the remote Kermit server to log itself out. You don't have to connect back to the remote host and clean up. However, if you want to connect back to the host, you can use the FINISH command instead of BYE, to shut down the Kermit server on the remote host without logging it off, allowing you to CONNECT back to your job there.

Here's an example of the use of a Kermit server. The user is sitting at a CP/M microcomputer and a DEC-20 is the remote host.

```
A>kermit                ! Run Kermit on the micro.
Kermit V3.2

Kermit-80>              ! This is the micro Kermit's prompt.
Kermit-80>connect       ! Connect to the DEC-20.
[Connecting to remote host.  Type CTRL-]C to return to micro.

CU20E                   ! The DEC-20 prints its herald.
@terminal vt52          ! Set your terminal type (optional).
@login my-id password                    ! Log in normally.
```

(The DEC-20 prints various login messages here.)

```
@kermit                 ! Run Kermit-20 normally
Kermit-20>server        ! Tell it to be a server.
[Kermit Server running on DEC-20 host.  Please type your esca
return to your local machine.  Shut down the server by typing
command on your local machine.]

^]c                     ! Now escape back to the micro.
[Connection closed, back at micro.]
Kermit-80>receive *.pas                  ! Get all my DEC-20
Kermit-80>send foo.*! Send all the "foo" files from my micro
Kermit-80>exit      .! Exit from Kermit back to CP/M.
A>
```

(Here you can do some work on the micro, edit files, whatever you like.)

```
A>kermit                ! Run Kermit-80 some more.
Kermit-80>send file.pas                  ! Send another file.
Kermit-80>bye           ! That's all.  Shut down the Kermit serv
A>                      ! Back at CP/M automatically.
```

This is much simpler. Note that once you've cranked up the Kermit Server on the remote end, you can run Kermit as often as you like on the micro without having to go back and forth any more; just make sure to shut the server down when you're done.

Here are the commands available for talking to servers. The underlined ones are standard.

SEND filespec — Sends a file or file group from the local host to the remote host in the normal way.

RECEIVE filespec — Ask the remote host to send a file or file group. Example:

```
receive *.c
```

This command is exactly equivalent to typing "send *.c" at the remote host followed by "receive" on the local host. Note that the local Kermit does not attempt parse the filespec. If the server cannot parse it, or cannot access the specified file(s), it will send back an appropriate error message.

BYE — Shut down the remote server and exit from Kermit. This will cause the job at the remote end to log itself out. You need not connect back and clean up unless you get an error message in response to this command (for instance, if your logged-out disk quota is exceeded on the remote host).

LOGOUT — Shut down the server but don't exit from Kermit.

FINISH — Shut down the server without having it log itself out, and don't exit from Kermit. A subsequent CONNECT command will put you back at your job on the remote host, at system command level.

If you want to run the server with non-default options selected (like IBM-FLAG, FILE-BYTE-SIZE, etc), then you can issue the appropriate SET options before giving the SERVER command.

2.7. When Things Go Wrong

There are various ways in which Kermit can become stuck, but since many hosts are capable of generating timeout interrupts when some process doesn't complete quickly enough, they can usually resend or "NAK" (negatively acknowledge) lost packets. Nevertheless, if a transfer seems to be stuck, you can type RETURN on the keyboard of most micros to wake up Kermit and have it retransmit the last packet.

An interesting exception is the IBM host (VM/CMS) Kermit -- it cannot time out its "virtual console" (i.e. the user's terminal), so when using Kermit from a micro to an IBM host, occasional manual wakeups may be necessary.

Here are a few symptoms and what to do about them.

2.7.1. The Transfer is Stuck

The following sections discuss various reasons why a transfer in progress could become stuck. Before examining these, first make sure that you really have a Kermit on the other end of the line, and you have issued the appropriate command: SEND, RECEIVE, or SERVER. If the remote side is not a server, remember that you must connect back between each transfer and issue a new SEND or RECEIVE command.

2.7.2. The Micro is Hung

The micro itself sometimes becomes hung for reasons beyond Kermit's control, such as power fluctuations. If the micro's screen has not been updated for a long time, then the micro may be hung. Try these steps (in the following order):

* Check the connection. Make sure no connectors have wiggled loose from their sockets. If you're using a modem, make sure you still have a carrier signal. Reestablish your connection if you have to.

* Press RETURN to wake the micro up. This should clear up any protocol deadlock.

* If the problem was not a deadlock, restart the micro and then restart Kermit, CONNECT back to the host, get back to your job or login again, and restart the transfer. You may have to stop and restart Kermit on the remote host.

2.7.3. The Remote Host Crashed

If your local system is working but the transfer is hung, maybe the remote host crashed. "Escape" back to local Kermit command level and issue the CONNECT command so that you can see what happened. If the remote system has crashed then you will have to wait for it to come back, and restart whatever file that was being transfered at the time.

2.7.4. The Disk is Full

If your floppy disk or directory fills up, the Kermit on the machine where this occurs will inform you and then terminate the transfer. You can continue the transfer by repeating the whole procedure either with a fresh floppy or after cleaning up your directory. If you were sending a file group from the DEC-20 you can continue the sequence where it left off by using the SEND command and including the name of the file that failed in the "(INITIAL)" field, for example

```
Kermit-20>send *.for (initial) foo.for
```

See the Kermit-20 command summary for further information about the initial filespec.

2.7.5. Host Errors

Various error conditions can occur on the remote host that could effect file transmission. Whenever any such error occurs, the remote Kermit normally attempts to send an informative error message to the local one, and then breaks transmission, putting you back at Kermit command level on the local system.

2.7.6. File is Garbage

There are certain conditions under which Kermit can believe it transferred a file correctly when in fact, it did not. One way this can happen is if errors occured during transmission that did not effect the byte count or checksum. For instance, if in a single packet one character bit n was received as a zero when it should have been a one, and in another character, bit n became a one when it should have been a zero, then the checksum would be the same as if no error had occured. The only remedies here are to edit the file after it is sent to fix the bad characters, or send the file again.

A more likely cause of garbage files has to do with the tricky business of the bytes. Most computers store files as sequences of 8-bit bytes, but there are some exceptions, like the DEC-10 and DEC-20, which store files in various ways -- text (printable) files as 7-bit bytes, binary files in 36-bit words. To complicate matters further, different systems handle telecommunication differently: most systems transmit ASCII (a 7-bit code) in 8-bit bytes, but some allow the application (user program) to use the 8th bit for data -- thus allowing the transmission of binary files in 8-bit bytes -- while others insist upon using the 8th bit as a parity check[11]. These complications only come into play when attempting to transfer binary files; when transferring ordinary text files, there's never a problem about byte sizes. The following descriptions of the particular Kermit implementations tell what mechanisms are available for dealing with bytes size problems.

---

[11]The parity bit is set to one or zero based upon whether sum of the other 7 bits is odd or even. Parity can be EVEN, ODD, MARK (always one), SPACE (always zero), or NONE (don't alter the 8th bit).